Journal of Engineering Science and Technology
Vol. 14, No. 2 (2019) 659 - 673
© School of Engineering, Taylor's University

SELECTION OF COMPONENTS USING RATE-OF-REUSABILITY AND MATRİX-OF-REUSABILITY

UMESH KUMAR TIWARI*, SANTOSH KUMAR

Department of Computer Science and Engineering, Graphic Era Deemed to be University, 566/6, Clement Town, Dehradun, Uttarakhand, India
*Corresponding Author: umeshtiwari22@gmail.com

Abstract

Reusability is the focal-point of component-based software engineering. This paper offers a selection measure for components, utilizing the reusability property of component-based applications. This work introduces a metric named 'Rate-of-Reusability', to explore and examine the reusability behaviour of various classes of components including fully-reusable and partially-reusable components. Further, Rate-of-Reusability is used to draw a 'Matrix-of-Reusability', which contains the reusability rate for defined categories of components. The Matrix-of-Reusability plays a vital role in the selection of components, specifically when we have more than one component available for reuse. Matrix-of-Reusability makes the selection process of components faster and more efficient. Values obtained through proposed metrics and matrix are stored as the component's attribute and can be used in future for developing component-based applications in a shorter period of time. Lines-of-code is used as the base metric for Rate-of-Reusability and Matrix-of-Reusability.

Keywords: Fully-reusable, Matrix-of-reusability, Partially-reusable, Rate-of-reusability, Reusability.

1. Introduction

In component-based software, we integrate different categories of components according to the design architecture and applications requirement. Efficient and regimented reuse of software components help to reduce the overall development period and cost of the component-based applications.

Software reusability is defined as the process of engineering new software products from the available software artefacts like structural design, architecture, code, code-lists, test data and test cases, instead of constructing them from the initial state [1-3]. Freeman [4] defined software reuse as, "the use of any information which a developer may need in the creation process". In Kruger [5] point of view, "reuse is the process of creating software systems from existing software rather than building them from scratch". Software reuse is the utilization of software components and systems that were build up in past for reuse in similar or heterogeneous contexts [6, 7]. In literature, reusability is defined for both categories of software constructs: modifiable and non-modifiable constructs [8-10]. Modifiable software constructs are those that can be reused with some minor or major changes in organized and controlled contexts [9, 10]. Non-modifiable constructs are the constructs that can be reused without making any change. Reusability is assumed as the extent of simplicity in reusing an artefact in a new perspective [11-13].

This work defines a method to estimate the reusability of components and to quantify the applicability of that component in new and underdevelopment component-based software applications.

Lines-of-code (LOC)

To define the Rate-of-Reusability, lines-of-code (LOC) of the component is used as the primary parameter. Line-of-code is one of the basic, recognizable and tangible size estimation metric [14]. Conte et al. [15] defined LOC as "any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements". In literature, two classes of LOCs are identified; Physical-LOC and Logical-LOC. Physical-LOCs are the count of statements including comment lines (though executable physical LOCs exclude the comments and blank spaces) whereas, the Logical-LOCs is the count of executable statements.

This work considers only the executable lines-of-code excluding blank spaces and comments. Our emphasis is on LOC rather than the individual code constructs.

1.1. Component-based software development (CBSD)

Component-based software development (CBSD) is one of the most promising and capable paradigms for the development of quality software systems. According to Tiwari and Kumar [16], CBSD is a selected branch of software development that offers building software products through available artefacts and constructs. These constructs are termed as 'components' in component-based development. Components interact with each other to share information. CBSD promotes reusing rather than regenerating. CBSD focuses on cooperation, coordination and integration of existing building blocks in an organized and defined manner. In CBSD the purpose is to develop a component (including classes, functions, methods or operations) only once and re-use them in various applications rather than being re-constructed every time. Reusing pre-developed and pre-tested components make the development life-cycle shorter, help to increase the reliability of the overall application, and reduce the time-to-market. Gaedke and Rehse [17] defined CBSD as “Component-Based Software Development aims at assembling large software systems from previously developed components (which in turn can be constructed from other components)”. The objective of component-based software engineering is to develop extremely large and complex software systems by integrating ‘Commercially Off-the-shelf Components (COTS)’, third-party contractual components and newly developed components to minimize the development time, and the cost [18, 19]. CBSD offers an improved and enhanced reuse of software components with additional properties including flexibility, extensibility and better service quality to discharge the needs of the end users [20-23].

1.2. Categories of components

Reusable components either are picked from the repository or are provided (or purchased) by some third party. On the basis of the amount of modifiability and the level of reuse, we identify three types of software components: Customized components (Major-customized components and Minor-customized components), Perfect components, and newly developed components [23-25].

1.2.1. Customized components

Customized components are those reusable components that require modifications in the code. On the basis of the level of modifications, customized components fall into two specific classes: Major-customized and Minor-customized components.

Major-customized components: These are the components, which can be reused in new contexts by introducing a wide range of changes in the code. Their reusability level is quite low, as they require major modifications. In such components, numbers of reused lines-of-code are comparatively less than new lines-of-code.

Minor-customized components: Minor-customized components require a lesser amount of changes to be reused in new application development. Their level of reusability is high in comparison to Major-customized components. Developers can reuse these components by making small changes in the component’s code. In such components, numbers of reused lines-of-code are comparatively greater than new lines-of-code.

1.2.2. Perfect components

These are the components that can be reused without any modifications in the code. They are picked from the repository and can be adapted without any change. Their reusability is of the highest degree.

1.2.3. Newly developed components

When there is no component in the repository that can be reused, new development takes place. Developers construct new components according to the requirements of the software under development.

2. Background Study

Prieto-Diaz and Freeman [26] identified one of the initial reusability measures in terms of size, structure, documentation, language and experience as the attributes of reusability. Authors employ lines-of-code for size estimation. To assess the design structure they used cyclomatic complexity, and to rate the documentation they proposed a rating scale of best (10) to worse (0). Caldiera and Basili [27] proposed expense, usability, and quality as the three aspects of reusability. Author's defined reusability through metrics like volume using Halstead Software Science, cyclomatic complexity using McCabe's method, regularity measures, and frequency of use.

Hristov et al. [28] classified reusability estimation metrics as the 'white-box' and the 'black-box' metrics. Metrics, which are used to assess the reusability of actual code, logic and structure of a component, are termed as white-box metrics. In spite of metrics, which are used to estimate the reusability of input/output, external interfaces and similar constructs, are called as black-box metrics.

Boxall and Araban [29] find in their study that the reuse level of a component is greatly affected by the component's understandability. They derived the value of understandability by using the attributes of the component's interfaces. In their work, Boxal and Arban take interface size, counts of the argument, number of repetitions, the scale of the repetitions and similar attributes to suggest some metrics for understandability. Washizaki et al. [30] suggested a Reusability-model for black-box components to reuse components efficiently. Authors identified some reusability affecting factors like functions offered, level of adaptation in new environments and varied requirements. The proposed metrics for better understandability of components. Gui and Scott [31] defined a metric-suite to compute the reusability of components and ranked them according to their reusability. These metrics are used to estimate the indirect coupling among components as well as to assess the degree of coupling.

Wijayasiriwardhane and Lai [32] suggested a size measurement technique named 'Component-point' to estimate the size of overall component-based systems. These component-points can be reused as a metric to analyse components for future use. They defined various classes of components according to their usability.

Chen and Lee [33] computed that to raise the efficiency of the reusability we have to decrease the values of program attributes like size, program volume, program level, difficulty to develop, and effort. Lee and Chang [34] defined complexity and modularity as the factors of reusability metrics. The proposed class complexity metrics for as inner and outer classes, and coupling and cohesion metrics for various classes. Cho et al. [35] proposed reusability metrics using the common functionality methods and the total number of available interface methods.

Majority of metrics available in the literature are defined either for conventional programs or for object-oriented software [36, 37]. There is a need to develop reusability assessment metrics for extremely large and complex software as component-based software. In this work, our attempt is to design such a metric that can be used to estimate not only the code constructs reusability but reusability of the components also.

3. Rate-of-Reusability Metric

Component-based software engineering integrates newly developed as well as reusable software components to engineering new software applications. The reciprocal relationship between the reusable and the newly developed components (ultimately the lines-of-code) define the Rate-of-Reusability of components. To define the Rate-of-Reusability, four types of Lines-of-Code (LOCs) are identified:

Total lines-of-code of a component (*$LOCC_i$*): Total LOC of a component is the sum of reused and the new LOCs.

Reused lines-of-code of a component (*$LOCC_{i\text{-}Reused}$*): Reused-LOCs comprise of Perfect and Customized LOCs of a component.

Customized lines-of-code of a component (*$LOCC_{i\text{-}Customozed}$*): Customized-LOCs consist of major and minor customized LOCs. Customized lines come under the category of Reused-LOCs. Customized lines require major or minor changes in the code.

New lines-of-code of a component (*LOCCi-New*): When existing LOCs fail to fulfil the customer's requirement, new LOCs are developed. In a component where we have the combination of reused and new LOCs, we can deduce New-LOCs by excluding Reused-LOCs from the total LOCs of the components.

3.1. Customized rate-of-reusability RoRCustomized

Customized components adapt available software constructs like code, test cases, test data, or documents with minor or major changes. On the basis of the rate of changes and modifications, we define two different scenarios: Major-customized RoR, and Minor-customized RoR.

3.1.1. Major-Customized rate-of-reusability *$RoRC_{i\text{-}Major\text{-}customized}$*

Major-customized components and LOCs require major modifications and alterations to be reused. At the component level, Major-customized reusability is computed in terms of LOCs of a particular component.

At the component level, Rate-of-Reusability has three cases -

Case 1: RoR when all types of LOCs are involved.

Case 2: RoR when only Reused-LOCs are involved.

Case 3: RoR when only Customized-LOCs are involved.

Case 1: RoR when all types of LOCs of the component are involved

This is the case when RoR metric is defined by considering the newly developed and Reused-LOCs of a particular component C_i. Here, RoR is computed by taking the ratio of Major-customized LOCs with the total LOCs of component C_i, as defined in Eq. (1).

$$RoRC_{i-Major-customized} = \frac{|LOCC_{i-Major-customized}|}{|LOCC_i|} \quad (1)$$

where $LOCC_i$ denotes the total LOCs of a component including new, customized and reused.

Case 2: Rate-of-reusability when only reused-LOCs of the component are involved

In this case, RoR is assessed by taking Reused (Customized and Perfect) LOCs of the component. Here we exclude the newly developed LOCs from the calculation. To compute the RoR, total numbers of Major-customized LOCs are divided by the total number of Reused-LOCs of the component *C_i*. It is defined in Eq. (2).

$$RoRC_{i-Major-customized} = \frac{|LOCC_{i-Major-customized}|}{|LOCC_{i-Reused}|} \quad (2)$$

where $LOCC_{i-Reused}$ denotes the collection of Perfect, Major-customized and Minor-customized reusable LOCs.

Case 3: RoR when only customized-LOCs of the component are involved

This is the case when the Rate-of-Reusability is calculated in the context of Customized-LOCs only. Total numbers of Major-customized LOCs are divided by the total numbers of Customized-LOCs of the component C_i, as shown in Eq. (3).

$$RoRC_{i-Major-customized} = \frac{|LOCC_{i-Major-customized}|}{|LOCC_{i-Customized}|} \quad (3)$$

where Customized-LOCs are the assembling of Major-customized and Minor-customized LOCs only.

3.1.2. Minor-customized rate-of-reusability $RoRC_{i\text{-}Minor\text{-}customized}$

Minor-customized components and LOCs require a minimum amount of modification to be reused. At the component level, reusability is computed considering the LOCs of particular components only.

There are three cases to compute the minor-customized Rate-of-Reusability-

Case 1: RoR when all types of LOCs are involved

Case 2: RoR when only Reused-LOCs are involved

Case 3: RoR when only Customized-LOCs are involved

Case 1: RoR when all categories of LOCs of the component are involved

In this context, the RoR metric is defined by considering newly developed and Reused-LOCs. Here, RoR is computed by taking the ratio of Minor-customized LOCs of a particular component C_i with the total number of LOCs of that component, as defined in Eq. (4).

$$RoRC_{i-Minor-customized} = \frac{|LOCC_{i-Minor-customized}|}{|LOCC_i|} \quad (4)$$

where $LOCC_i$ denotes the total LOCs of a component including new, customized and reused.

Case 2: RoR when only reused-LOCs of the component are involved

Here, RoR is assessed by taking the Reused (Customized and Perfect) LOCs of the component. In this case, we exclude the newly developed LOCs from the calculation. To compute the RoR, total numbers of Minor-customized LOCs are divided by the total number of reused LOCs of the component, as defined in Eq. (5).

$$RoRC_{i-Minor-customized} = \frac{|LOCC_{i-Minor-customized}|}{|LOCC_{i-Reused}|} \quad (5)$$

where $LOCC_{i-Reused}$ denotes the collection of Perfect, Major-customized and Minor-customized LOCs.

Case 3: RoR when only customized-LOCs of the component are involved

In this case, Rate-of-Reusability is taken by considering Customized-LOCs only. Total numbers of Minor-customized LOCs are divided by the total Customized-LOCs of the component, as defined in Eq. (6).

$$RoRC_{i-Minor-customized} = \frac{|LOCC_{i-Minor-customized}|}{|LOCC_{i-Customized}|} \quad (6)$$

where Customized-LOCs are the assembling of Major-customized and Minor-customized LOCs only.

3.2. Perfect rate-of-reusability RoRPerfect

At the individual component level we include all categories of components and LOCs including New, Reused and Customized. We have three cases:

Case 1: RoR when all categories of LOCs are involved.

Case 2: RoR when only Reused-LOCs are involved.

Case 3: RoR when only Customized-LOCs are involved.

Case 1: RoR when all categories of LOCs of the component are involved

In this case, RoR metric is defined by considering the newly developed and Reused-LOCs. Here, RoR is computed by taking the ratio of Perfect-LOCs with total LOCs of the component, as defined in Eq. (7).

$$RoRC_{i-Perfect} = \frac{|LOCC_{i-Perfect}|}{|LOCC_{i}|} \quad (7)$$

where $\mathrm{RoRC_{i-Perfect}}$ is the Rate-of-Reusability metric at the component level, $\mathrm{LOCC_{i-Perfect}}$ denotes the number of LOCs in the component, which can be reused without any change, and $\mathrm{LOCC_i}$ denotes the total number of LOCs in the component.

Case 2: RoR when only reused LOCs of the component are involved

In this case, to calculate the RoR, total numbers of Perfect-LOCs are divided by the total number of reused LOCs of the component, as defined in Eq. (8).

$$RoRC_{i-Perfect} = \frac{|LOCC_{i-Perfect}|}{|LOCC_{i-Reused}|} \quad (8)$$

Case 3: RoR when only Customized-LOCs of the component are involved

In this case, the Rate-of-Reusability is taken in the context of Customized-LOCs only. Total number of Perfect-LOCs is divided by the total numbers of Customized-LOCs of the component, as defined in Eq. (9).

$$RoRC_{i-Perfect} = \frac{|LOCC_{i-Perfect}|}{|LOCC_{i-Customized}|} \quad (9)$$

where Customized-LOCs are the assembling of Major-customized and Minor-customized LOCs only.

4. Matrix-of-Reusability (MoR)

Matrix-of-Reusability (MoR) is defined as a row-column matrix containing the values of Rate-of-Reusability of various components. Matrix-of-Reusability stores the values of RoR of each candidate component and is used to select the appropriate component. Rate-of-Reusability for different categories of components and different types of LOCs is represented in row headers and column headers show the different available components having similar functionality, as shown in Table 1. Matrix-of-Reusability not only stores the values of RoR of components but is also used to select appropriate and most reusable component.

We use RoR metric to identify a reusable component when we have more than one component choices. With the help of RoR of candidate components (components providing similar functionality), first, we draw the Matrix-of-Reusability. Now, the component having the highest Rate-of-Reusability value is selected as the reusable component in software underdevelopment.

Table 1. Matrix-of-reusability.

	Candidate Components			
MoR	**Component C_1**	**Component C_2**	**Component C_3**	**Component C_4**
$RoRC_{i\text{-}Major\text{-}Customized}$	Value of major-customized RoR of C_1	Value of major-customized RoR of C_2	Value of major-customized RoR of C_3	Value of major-customized RoR of C_4
$RoRC_{i\text{-}Minor\text{-}Customized}$	Value of minor-customized RoR of C_1	Value of minor-customized RoR of C_2	Value of minor-Customized RoR of C_3	Value of minor-customized RoR of C_4
$RoRC_{i\text{-}Perfect}$	Value of perfect RoR of C_1	Value of perfect RoR of C_2	Value of perfect RoR of C_3	Value of perfect RoR of C_4

5. Case Study

To show our experiments, we have considered a case study of a part of web based software.

Figure 1 shows the design structure of the ‘Login Process’ of a web page having four components LoginPage, LoginVerification, LoginDatabase and UserHomePage. User fills login information in LoginPage, which is verified by LoginVerification component. This verification is done with the help of user LoginDatabase component. If the user data is verified then the UserHomePage component must be active, otherwise, LoginVerification redirects the LoginPage component. Here the aim is to select a suitable UserHomePage (shown in oval) component. In repository we have four similar intention candidate components denoted as UHP_1, UHP_2, UHP_3, and UHP_4, where UHP denotes UserHomePage.

We have four components for the same purpose and we have to select one among them for reuse in under-development software as shown in Table 2. LOCs of these components are available with each candidate component. Total LOCs of candidate component UHP_1 are 350, UHP_2 have 320 LOCs, UHP_3 have 380, and UHP_4 have 370 LOCs as shown in Table 2.

Now out of these LOCs, the developer can identify the Major-customized, Minor-customized and Perfect-LOCs of the component. In this case study, we have values of Major-customized, Minor-customized and Perfect-LOCs of the component as shown in Table 2. Here columns represent the candidate component's name and the rows represent the LOCs of the corresponding components. In this case study, we have considered the selection procedure at the component level only.

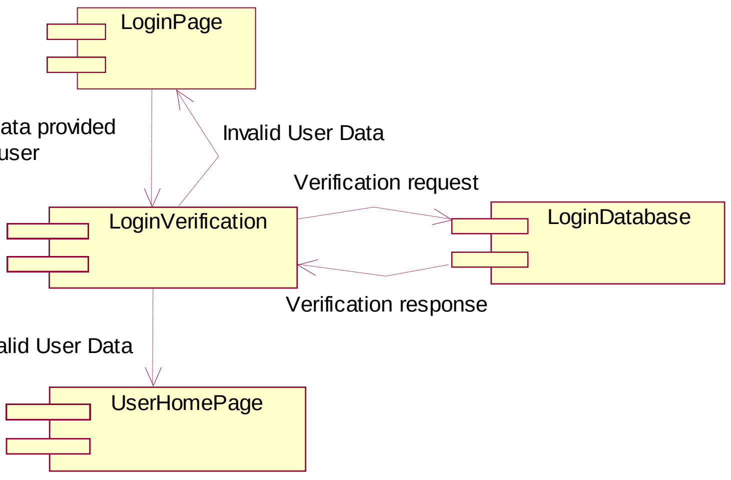


Fig. 1. Case study of login process.

RoRs of login process

Now from Table 2, we can compute the values of the number of Reused, Customized and New-LOCs of the candidate components. Respective computed values are shown in Table 3.

With the help of Tables 2 and 3, now we can draw the Matrix-of-Reusability for four candidate components using the Rate-of-Reusability metric defined in Eqs. (1) to (9).

Table 2. Major-customized, minor-customized and perfect-LOCs of candidate components.

Lines of Code	UHP_1	UHP_2	UHP_3	UHP_4
$\lvert LOCC_i \rvert$	350	320	380	370
$\lvert LOCC_{i\text{-Major-customized}} \rvert$	120	120	100	150
$\lvert LOCC_{i\text{-Minor-customizedl}} \rvert$	90	30	100	120
$\lvert LOCC_{i\text{-Perfect}} \rvert$	55	60	30	60

Table 3. Reused customized and new-LOCs of 4 candidate components.

Lines of code	UHP_1	UHP_2	UHP_3	UHP_4
LOC_{Reused}	265	210	230	330
$LOC_{Customized}$	210	150	200	270
LOC_{New}	85	110	150	40

(a) MoR and selection of components when all types of LOCs of the component are involved in the computation

Applying the values given in Tables 2 and 3 on equations proposed for reusability metric, we assess the values of Rate-of-Reusability metric of candidate component, in the context when calculations are made in terms of all parts of the component, that is, new and reused.

Values of Tables 2 and 3 are applied on equations defined above to achieve the $RoRC_i$ of four candidate components. To compute the $RoRC_{i\text{-}Major\text{-}customized}$ of four candidate components, we apply the values of Tables 1 and 2 in Eq. (1). To compute the $RoRC_{i\text{-}Minor\text{-}customized}$ and $RoRC_{i\text{-}Perfect}$ of candidate components UHP$_1$, UHP$_2$, UHP$_3$, and UHP$_4$ we apply the values of Tables 2 and 3 on Eqs. (4) and (7) respectively. These computations are shown in Fig. 2.

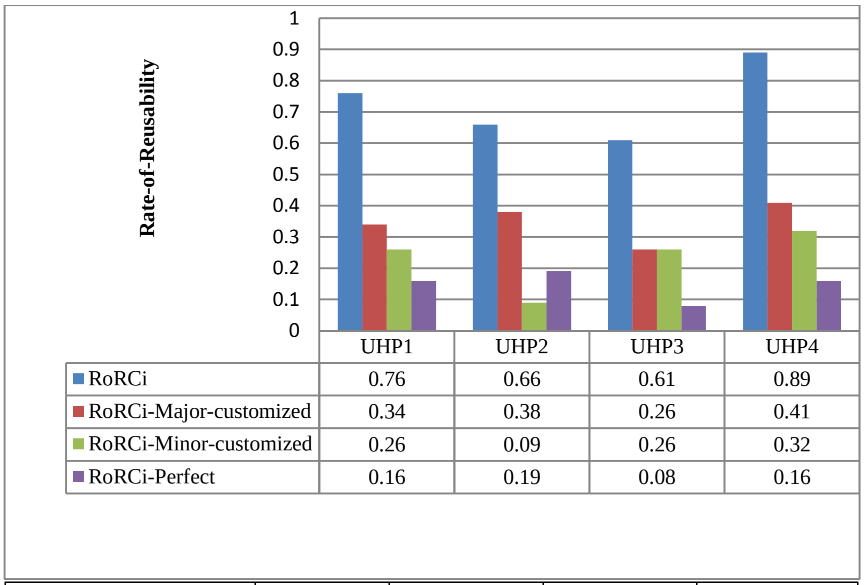
From Fig. 2 we note that, when selection is made at the component level, then the eligible components are:

Maximum Rate-of-Reusability value: Component UHP$_4$.

Maximum Major-customized value: Component UHP$_4$.

Maximum Minor-customized value: Component UHP$_4$.

Maximum Perfect value: Component UHP$_2$.



	UHP1	UHP2	UHP3	UHP4
🟦 RoRCi	0.76	0.66	0.61	0.89
🟥 RoRCi-Major-customized	0.34	0.38	0.26	0.41
🟩 RoRCi-Minor-customized	0.26	0.09	0.26	0.32
🟪 RoRCi-Perfect	0.16	0.19	0.08	0.16

Total RoR of all LOCs	0.76	0.66	0.60	**0.89**

Fig. 2. Reusability-graph when all LOCs of component are involved.

(b) MoR and selection of components when only reused-LOCs of the component are involved in the computation

Now using the Rate-of-Reusability metric method and Tables 2 and 3, we calculate the values of Rate-of-Reusability in the context of Reused-LOCs only and draw the Matrix-of-Reusability as shown in Fig. 3.

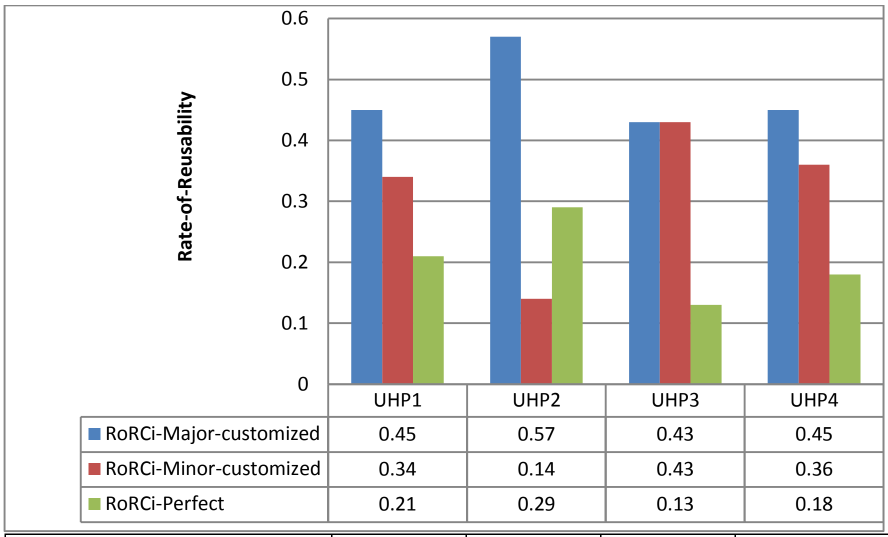
Values of Tables 2 and 3 are applied on Eq. (2) to achieve the RoRCi-Major-custom$_{\text{ized}}$ of four candidate components. We apply the values of Tables 2 and 3 in Eqs. (5) and (8) to compute the $RoRC_{i\text{-Minor-customized}}$ and $RoRC_{i\text{-Perfect}}$ of candidate components UHP$_1$, UHP$_2$, UHP$_3$, and UHP$_4$ respectively. These computations are shown in Fig. 3.

From Fig. 3, we note that when selection is made at the reused component level, then the eligible components are:

Maximum major-customized value: Component UHP2.

Maximum Minor-customized value: Component UHP3.

Maximum Perfect value: Component UHP2.



	UHP1	UHP2	UHP3	UHP4
🟦 RoRCi-Major-customized	0.45	0.57	0.43	0.45
🟥 RoRCi-Minor-customized	0.34	0.14	0.43	0.36
🟩 RoRCi-Perfect	0.21	0.29	0.13	0.18
Total RoR of Reused- LOCs	**1**	**1**	0.99	0.99

Fig. 3. Reusability-graph when only reused-LOCs are involved.

(c) MoR and selection of components when only customized-LOCs of the component are involved in the computation

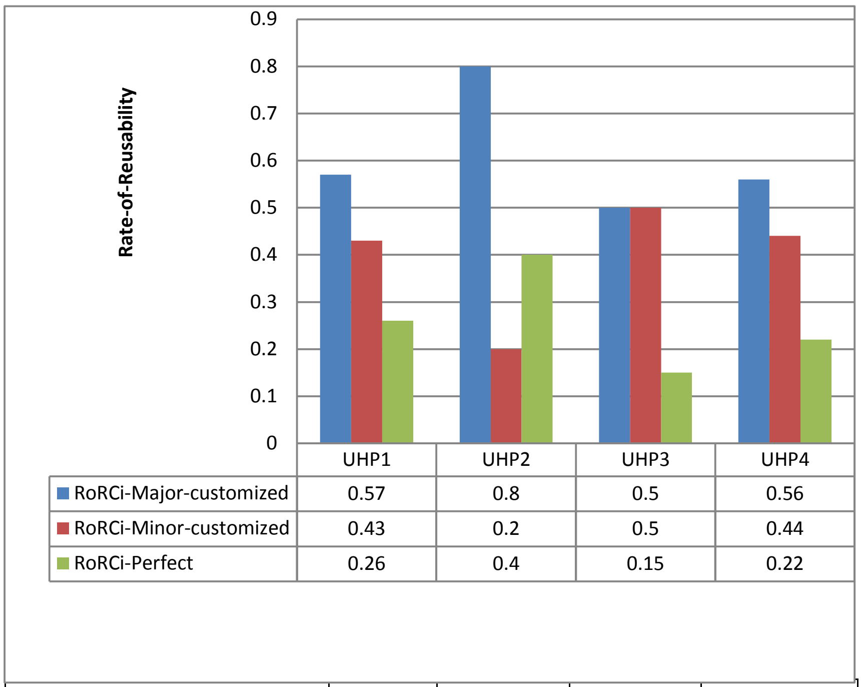
We can estimate the Rate-of-Reusability metric in the context of Customized-LOCs only, with the help of Tables 2 and 3. Values of Tables 2 and 3 are applied on Eq. (3) to achieve the $RoRC_{i\text{-Major-customized}}$ of four candidate components. We apply the values of Tables 2 and 3 in Eqs. (6) and (9) to compute the $RoRC_{i\text{-Minor-customized}}$ and $RoRC_{i\text{-Perfect}}$ of candidate components UHP$_1$, UHP$_2$, UHP$_3$, and UHP$_4$ respectively. These computations are shown in Fig. 4.

From Fig. 4, we note that when selection is made at the adaptable component level, then the eligible components are:

Maximum Major-customized value: Component UHP$_2$.

Maximum Minor-customized value: Component UHP$_3$.

Maximum Perfect value: Component UHP$_4$.



	UHP1	UHP2	UHP3	UHP4
🟦 RoRCi-Major-customized	0.57	0.8	0.5	0.56
🟥 RoRCi-Minor-customized	0.43	0.2	0.5	0.44
🟩 RoRCi-Perfect	0.26	0.4	0.15	0.22

Total RoR of customized-LOCs	**1.26**	1.4	1.15	**1.22**

Fig. 4. Reusability-graph when only customized-LOCs are involved.

6. Conclusion

Analytical Aim of this work is to facilitate developers in the selection of appropriate components, in minimum time. Measures and metrics proposed and used so far in literature are based on either conventional programming or object-oriented software. In the context of reusability, researchers used programming constructs like operators and operands rather than the whole LOC. In our calculations, we have identified three classes of components and lines-of-code namely, New, Customized, and Perfect. We define the Rate-of-Reusability metric for these identified components and lines-of-code at the system level and at the individual component level. With the help of Rate-of-Reusability, we compute the Matrix-of-Reusability for a number of candidate components serving the same purpose. In our study, we have used a case study of a login process containing four components and concluded by three conditions as:

- When new and reused lines-of-code are involved in the component selection then the component having RoR value greater than 0.75 and less than or equal to 1(that is, $0.75 \leq \text{RoR} \leq 1$) is eligible for reuse in the current software under development.
- When only reused that is, Major-customized, Minor-customized and Perfect lines-of-code are involved in the selection then the component having RoR value greater than 0.9 and less than or equal 1 (that is, $0.9 \leq \text{RoR} \leq 1$) can be reused.

- When only Customized components are involved in the selection then the component having RoR value greater than 1.5 and less than or equal to 2 (that is, $1.5 \leq \text{RoR} \leq 2$) can be reused.

The obtained results conclude that the increase in the reusability of a component leads in the increase in selection probability of that component in the component-based software.

References

1. Johnson, W.L.; and Harris, D.R. (1991). Sharing and reuse of requirements knowledge. *Proceedings of the 6th Annual Conference on Knowledge-Based Software Engineering Conference.* Syracuse, New York, United States of America, 57-66.
2. Maiden, N.; and Sutcliffe, A. (1991). Analogical matching for specification reuse. *Proceedings of 6th Annual Conference on Knowledge-Based Software Engineering (KBSE).* Syracuse, New York, United States of America, 108-116.
3. Davis, A.M.; and Bersoff, E.F. (1991). Impacts of life cycle models on software configuration management. *Communications of the ACM*, 34(8), 104-118.
4. Freeman, P. (1983). Reusable software engineering: Concepts and research directions. *Proceedings of the ITT Workshop on Reusability in Programming.* Newport, Rhode Island, England, 129-137.
5. Kruger, C.W. (1995). Software reuse. *ACM Computing Surveys (CSUR)*, 24(2), 131-183.
6. Tracz, W. (1995). *Confessions of a used program salesman: Institutionalizing software reuse* (1st ed.). Boston, Massachusetts, United States of America: Addison-Wesley.
7. Christine, L.B. (1994). Reuse. *Marciniak*, 1055-1069.
8. Lim, W.C. (1994). Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5), 23-30.
9. Mcilroy, M.D. (1968). Mass produced software components. *Proceedings of the NATO Conference on Software Engineering. Garmisch, Germany*, 88-98.
10. Cooper, J. (1994). Reuse-the business implications. *Marciniak*, 1071-1077.
11. Prieto-Diaz, R. (1989). Classification of reusable modules. *Software Reusability: Concepts and Models*, 1, 99-123.
12. Cybulski, J.L. (1996). Introduction to software reuse. Department of Information Systems, University of Melbourne, Melbourne, Australia. *Technical Report TR 96/4.*
13. Kim, Y.; and Stohr, E.A. (1998). Software reuse: Survey and research directions. *Journal of Management Information Systems*, 14(4), 113-147.
14. Zhao, Y.; Yang, Y.; Xu, B.; Leung, H.; and Zhou, X. (2014). Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study. *Information and Software Technology*, 56(2), 220-237.
15. Conte, S.D.; Dunsmore, H.E.; and Shen, V.Y. (1986). *Software engineering metrics and models*. Redwood City, California: Benjamin-Cummings Publishing Co. Inc.

16. Tiwari, U.; and Kumar, S. (2014). Cyclomatic complexity metric for component based software. *ACM SIGSOFT Software Engineering Notes*, 39(1), 1-6.
17. Gaedke, M.; and Rehse, J. (2000). Supporting compositional reuse in component-based web engineering. *Proceedings of the ACM Symposium on Applied Computing (Volume 2).* New York, United States of America, 927-933.
18. Boehm, B.W.; Pendo, M.; Pyster, A.; Stuckle, E.D.; and William, R.D. (1984). An environment for improving software productivity. *IEEE Computer,* 12.
19. Brereton. B.; and Budgen, D. (2000). Component-based systems: A classification of issues. *IEEE Computer*, 33(11), 54-62.
20. Vitharana, P.; Zahedi, F.M.; and Jain, M. (2003). Design, retrieval and assembly in component-based software development. *Communications of the ACM*, 46(11), 97-102.
21. Basili, V.R.; and Boehm, B. (2001). Cots-based systems top 10 list. *Computer,* 34(5), 91-95.
22. Chen, J.; Wang, H.; Zhou, Y.; and Bruda, D.S. (2001). Complexity metrics for component-based software systems. *International Journal of Digital Content Technology and its Applications*, 5(3), 235-244.
23. Tyaki, K.; and Sharma, A. (2012). Reliability of component based systems - A critical survey. *ACM SIGSOFT Software Engineering Notes*, 36(6), 1-6.
24. Tiwari, U.K.; and Santosh, K. (2016). Components integration-effect graph: A black box testing and test case generation technique for component-based software. *International Journal of Systems Assurance Engineering and Management,* 8(2), 393-407.
25. Bennatan, E.M. (1992). *Software project management: A Practitioner's Approach.* London, United Kingdom: McGraw-Hill.
26. Prieto-Diaz, R.; and Freeman, P. (1987). Classifying software for reusability. *IEEE Software,* 4(1), 6-16.
27. Caldiera, G.; and Basili, V.R. (1991). Identifying and qualifying reusable software components. *Computer*, 24(2), 61-70.
28. Hristov, D.; Hummel, O.; Huq, M.; and Janjic, W. (2012). Structuring software reusability metrics for component-based software development. *Proceedings of the Seventh International Conference on Software Engineering Advances (ICSEA).* Lisbon, Portugal, 421-429.
29. Boxall, M.A.S.; and Araban, S. (2004). Interface metrics for reusability analysis of components. *Proceedings of the Conference on Australian Software Engineering (ASWEC).* Melbourne, Victoria, Australia, 40-51.
30. Washizaki, H.; Yamamoto, H.; and Fukazawa, Y. (2003). A metrics suite for measuring reusability of software components. *Proceedings of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry.* Sydney, New South Wales, Australia, 211-223.
31. Gui, G.; and Scott, P.D. (2007). Ranking reusability of software components using coupling metrics. *The Journal of Systems and Software*, 80(9), 1450-1459.

32. Wijayasiriwardhane, T.; and Lai, R. (2010). Component point: A system-level size measure for component-based software systems. *Journal of Systems and Software*, 83(12), 2456-2470.
33. Chen, D.-J.; and Lee, P.J. (1993). On the study of software reuse using reusable C++ components. *Journal of Systems and Software,* 20(1), 19-36.
34. Lee, Y.; and Chang, K.H. (2000). Reusability and maintainability metrics for object-oriented software. *Proceedings of the 38th Annual South East Regional Conference*. Clemson, South Carolina, United States of America, 88-94.
35. Cho, E.S.; Kim, M.S.; and Kim, S.D. (2001). Component metrics to measure component quality. *Proceedings of the Eighth Asia-Pacific Conference on Software Engineering*. Macao, China, 419-426.
36. Nunez-Varela, A.S.; Perez-Gonzalez, H.G.; Martinez-Perez, F.E.; and Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. *Journal of Systems and Software*, 128, 164-197.
37. Suresh, Y.; Pati, J.; Santanu, K.R.; and Rath, S.K. (2012). Effectiveness of software metrics for object-oriented system. *Procedia Technology*, 6, 420-427.